Evaluating location predictors with extensive Wi-Fi mobility data

Libo Song, David Kotz
Dartmouth College
6211 Sudikoff, Hanover, NH 03755
{lsong,dfk}@cs.dartmouth.edu

Ravi Jain, Xiaoning He
DoCoMo USA Labs
181 Metro Drive, San Jose, CA 95110
{jain,xiaoning}@docomolabs-usa.com

Abstract— **Location is an important feature for many applications, and wireless networks can better serve their clients by anticipating client mobility. As a result, many location predictors have been proposed in the literature, though few have been evaluated with empirical evidence. This paper reports on the results of the first extensive empirical evaluation of location predictors, using a two-year trace of the mobility patterns of over 6,000 users on Dartmouth's campus-wide Wi-Fi wireless network.**

We implemented and compared the prediction accuracy of several location predictors drawn from two major families of domain-independent predictors, namely Markov-based and compression-based predictors. We found that low-order Markov predictors performed as well or better than the more complex and more space-consuming compression-based predictors. Predictors of both families fail to make a prediction when the recent context has not been previously seen. To overcome this drawback, we added a simple fallback feature to each predictor and found that it significantly enhanced its accuracy in exchange for modest effort. Thus the Order-2 Markov predictor with fallback was the best predictor we studied, obtaining a median accuracy of about 72% for users with long trace lengths. We also investigated a simplification of the Markov predictors, where the prediction is based not on the most frequently seen context in the past, but the most recent, resulting in significant space and computational savings. We found that Markov predictors with this recency semantics can rival the accuracy of standard Markov predictors in some cases. Finally, we considered several seemingly obvious enhancements, such as smarter tie-breaking and aging of context information, and discovered that they had little effect on accuracy. The paper ends with a discussion and suggestions for further work.

I. Introduction

A fundamental problem in mobile computing and wireless networks is the ability to track and predict the location of mobile devices. An accurate location predictor can significantly improve the performance or reliability of wireless network protocols, the wireless network infrastructure itself, and many applications in pervasive computing. These improvements lead to a better user experience, to a more cost-effective infrastructure, or both.

For example, in wireless networks the SC (Shadow Cluster) scheme can provide consistent QoS support level before and after handover [1]. In the SC scheme, all cells neighboring a mobile node's current cell are requested to reserve resources for that node, in case it moves to that cell next. Clearly, this approach ties up more resources than necessary. Several proposed SC variations [2] attempt to predict the device's movement so the network can reserve resources in only certain neighboring cells.

Location prediction has been proposed in many other areas of wireless cellular networks as a means of enhancing performance, including better mobility management [3], improved assignment of cells to location areas [4], more efficient paging [5], and call admission control [6].

Many pervasive computing applications include opportunities for location-based services and information. With a location predictor, these applications can provide services or information based on the user's next location. For example, consider a university student that moves from dormitory to classroom to dining hall to library. When the student snaps a photo of his classmates using a wireless digital camera and wishes to print it, the camera's printing application might suggest printers near the current location and near the next predicted location, enabling the photo to finish printing while the student walks across campus.

Both of these examples depend on a predictor that can predict the next wireless cell visited by a mobile device, given both recent and long-term historical information about that device's movements. Since an accurate prediction scheme is beneficial to many applications, many such predictors have been proposed, and recently surveyed by Cheng et al. [7].

To the best of our knowledge, no other researchers have evaluated location predictors with extensive mobility data from real users. Das et al. use two small user mobility data sets to verify the performance of their own prediction schemes, involving a total of five users [8].

In this paper we compare the most significant domain-independent predictors using a large set of user mobility data collected at Dartmouth College. In this data set, we recorded for two years the sequence of wireless cells (Wi-Fi access points) frequented by more than 6000 users.

After providing more background on the predictors and our accuracy metric, we present the detailed results we obtained by running each predictor over each user's trace. We found that even the simplest classical predictors can obtain a median prediction accuracy of about 72% over all users with sufficiently long location histories (1000 cell crossings or more), although accuracy varied widely from user to user. This performance may be sufficient for many useful applications. On the other hand, many applications will need higher accuracy.

Although predicting the time of the occurrence of next movement besides the next location is important for some applications, in this paper we focus on predicting the next location, since most domain-independent predictors do not predict time.

We found that the simple Markov predictors performed as well or better than the more complicated LZ predictors, with smaller data structures. We also found that many predictors often fail to make any prediction, but our simple fallback technique provides a prediction and improves overall accuracy. We conclude the paper with a more extensive summary of our conclusions.

II. BACKGROUND

In this section, we discuss the nature of location prediction, and summarize several predictors from the literature. We define the metrics we used to evaluate the predictors, and how we collected the empirical data set we used in our evaluation.

A. Location

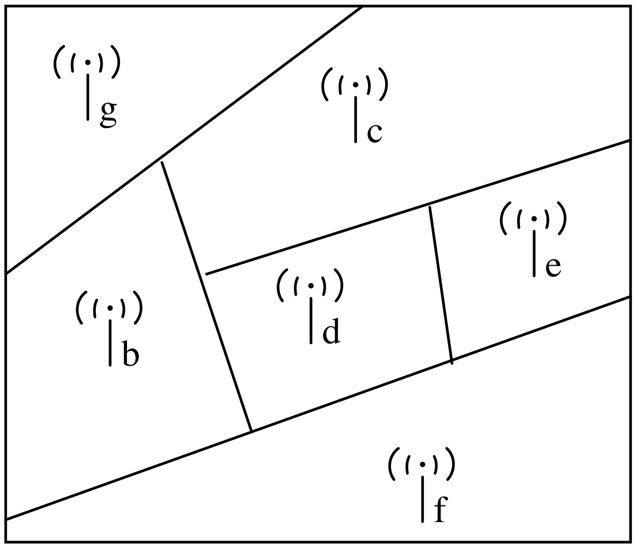
In the context of this work we assume that a user resides at a given discrete location at any given time; sample locations include "room 116" within a building, "Thayer Dining Hall" within a campus, "cell 24" within a cellular network, or "berry1-ap" access point within an 802.11 wireless network. (In our data, as we discuss below, each location is the name of an access point with which the user's device associated.) We list all possible such locations in a finite alphabet $\mathcal{A}$, and can identify any location as a symbol a drawn from that alphabet. For a given user we list the sequence of locations visited, its *location history* L, as a string of symbols. If the history has n locations, $L_n = a_1 a_2 \dots a_n$.

The location history may be a sequence of location *observations*, for example, the user's location recorded once every five seconds, or a sequence of location *changes*. In the latter case, $a_i \neq a_{i+1}$ for all $0 < i < n$. All of the predictors we consider in this paper are agnostic to this issue. It happens that our data is a sequence of location changes.

All of the predictors we consider in this paper are domain-independent and operate on the string L as a sequence of abstract symbols. They do not place any interpretation on the symbols. For that reason, our location history does not include any timing information, or require any associated information relating the symbols such as geographic coordinates. As an example, though, consider the environment with six wireless cells (labeled b through g) diagrammed in Figure 1, accompanied by one possible location history.

B. Domain-independent predictors

In this paper we consider only domain-independent predictors; we will examine domain-dependent predictors in future work. We are interested in *on-line predictors*, which examine the history so far, extract the current context, and predict the next location. Once the next location is known, the history is



Sample location history $L = gbdcbgcefbdbde$

Figure 1. Sample cell map and location history

now one symbol longer, and the predictor updates its internal tables in preparation for the next prediction.

During the rest of this section, we discuss two families of domain-independent predictors, Order-k Markov predictors and LZ-based predictors.

Markov family

The order-k (or "$O(k)$") Markov predictor assumes that the location can be predicted from the current *context*, that is, the sequence of the k most recent symbols in the location history $(a_{n-k+1}, \dots, a_n)$. The underlying Markov model represents each state as a context, and transitions represent the possible locations that follow that context.

Consider a user whose location history is $L = a_1 a_2 \dots a_n$. Let substring $L(i,j) = a_i a_{i+1} \dots a_j$ for any $1 \leq i \leq j \leq n$. We think of the user's location as a random variable X. Let $X(i,j)$ be a string $X_i X_{i+1} \dots X_j$ representing the sequence of random variates $X_i, X_{i+1}, \dots X_j$ for any $1 \leq i \leq j \leq n$. Define the context $c = L(n-k+1, n)$. Let $\mathcal{A}$ be the set of all possible locations. The Markov assumption is that X behaves as follows, for all $a \in \mathcal{A}$ and $i \in \{1, 2, \dots, n\}$.

$$\begin{aligned} &P(X_{n+1} = a | X(1,n) = L) \\ &\quad = P(X_{n+1} = a | X(n-k+1, n) = c) \\ &\quad = P(X_{i+k+1} = a | X(i+1, i+k) = c) \end{aligned}$$

where the notation $P(X_i = a_i | \dots)$ denotes the probability that X_i takes the value a_i. The first two lines indicate the assumption that the probability depends only on the context of the k most recent locations. The latter two lines indicate the assumption of a stationary distribution, that is, that the probability is the same anywhere the context is the same.

These probabilities can be represented by a *transition probability matrix* M. Both the rows and columns of M are indexed by length-k strings from $\mathcal{A}^k$ so that $P(X_{n+1} = a|X(1,n) = L(1,n)) = M(s,s')$ where $s = L(n-k+1,n)$, the current context, and and $s' = L(n-k+2,n)a$, the next context. In that case, knowing M would immediately provide the probability for each possible next symbol of L.

Since we do not know M, we can generate an estimate $\widehat{P}$ from the current history L, the current context c, and the equation

$$\widehat{P}(X_{n+1} = a|L) = \frac{N(ca,L)}{N(c,L)} \tag{1}$$

where $N(s',s)$ denotes the number of times the substring s' occurs in the string s.

Given this estimate, we can easily define the behavior of the $O(k)$ Markov predictor. It predicts the symbol $a \in \mathcal{A}$ with the maximum probability $\widehat{P}(X_{n+1} = a|L)$, that is, the symbol that most frequently followed the current context c in prior occurrences in the history. Notice that if c has never occurred before, the above equation evaluates to $0/1 = 0$ for all a, and the $O(k)$ Markov predictor makes no prediction.

If the location history is not generated by an order-k Markov source, then this predictor is, of course, only an approximation.

Fortunately, $O(k)$ Markov predictors are easy to implement. Our implementation maintains an estimate of the (sparse) matrix M, using Equation 1. To make a prediction, the predictor scans the row of M corresponding to the current context c, choosing the entry with the highest probability for its prediction. After the next move occurs, the predictor updates the appropriate entry in that row of M, and updates c, in preparation for the next prediction.

Vitter and Krishnan [9] use $O(k)$ predictors to prefetch disk pages, and prove interesting asymptotic properties of these predictors. Other variations of Markov predictors can be found in the survey [7].

LZ family

LZ-based predictors are based on a popular incremental parsing algorithm by Ziv and Lempel [10] often used for text compression. This approach seems promising because (a) most good text compressors are good predictors [9] and (b) LZ-based predictors are like the $O(k)$ Markov predictor except that k is a variable allowed to grow to infinity [5]. We briefly discuss how the LZ algorithm works.

Let γ be the empty string. Given an input string s, the LZ parsing algorithm partitions the string into distinct substrings $s_0, s_1, \ldots, s_m$ such that $s_0 = \gamma$, and for all $j > 0$ substring s_j without its last character is equal to some $s_i, 0 \le i < j$, and $s_0 s_1 \ldots s_m = s$. Observe that the partitioning is done sequentially, i.e., after determining each s_i the algorithm only considers the remainder of the input string. Using the example from Figure 1, $s = gbdcbgcefbdbde$ is parsed as $\gamma, g, b, d, c, bg, ce, f, bd, bde$.

Associated with the algorithm is a tree, the *LZ tree*, that is grown dynamically during the parsing process. Each node of the tree represents one substring s_i. The root is labeled γ and the other nodes are labeled with a symbol from $\mathcal{A}$ so that the sequence of symbols encountered on the path from the root to that node forms the substring associated with that node. Since this LZ tree is to be used for prediction, it is necessary to store some statistics at each node. The tree resulting from parsing the above example is shown in Figure 2; each node is labeled with its location symbol and the value of its statistic, a counter, after parsing.

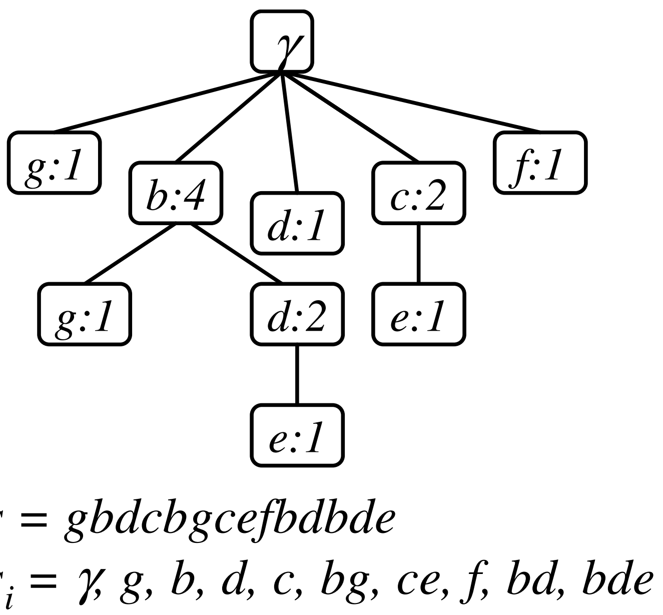


Figure 2. Example LZ parsing tree

To parse a (sub)string s we trace a path through the LZ tree. If any child of the current node (initially the root) matches the first symbol of s, remove that symbol from s and step down to that child, incrementing its counter; continue from that node, examining the next symbol from s. If the symbol did not match any child of the current node, then remove that symbol from s and add a new child to the current node, labeled with that symbol and counter=1; resume parsing at the root, with the now shorter string s.

Based on the LZ parsing algorithm, several predictors have been suggested in the past [9], [11], [12], [5], [6]. We describe some of these below and then discuss how they differ. For more detailed information, please refer to [7].

LZ predictors. Vitter and Krishnan [9] considered the case when the generator of L is a finite-state Markov source, which produces sequences where the next symbol is dependent on only its *current state*. (We note that a finite-state Markov source is different from the $O(k)$ Markov source in that the states do not have to correspond to strings of a fixed length from $\mathcal{A}$.) They suggested estimating the probability, for each $a \in \mathcal{A}$, as

$$\widehat{P}(X_{n+1} = a|L) = \frac{N^{LZ}(s_m a, L)}{N^{LZ}(s_m, L)} \tag{2}$$

where $N^{LZ}(s', s)$ denotes the number of times s' occurs as a prefix among the substrings $s_0, \ldots, s_m$ which were obtained by parsing s using the LZ algorithm.

It is worthwhile comparing Equation (1) with Equation (2). While the former considers how often the string of interest occurs in the entire input string, the latter considers how often it occurs in the partitions s_i created by LZ. Thus, in the example of Figure 2, while *dc* occurs in L it does not occur in any s_i.

The LZ predictor chooses the symbol a in $\mathcal{A}$ that has the highest probability estimate, that is, the highest value of $\widehat{P}(X_{n+1} = a|L)$. An on-line implementation need only build the LZ tree as it parses the string, maintaining node counters as described above. After parsing the current symbol the algorithm rests at a node in the tree. It examines the children of the current node, if any, and predicts the symbol labeling the child with the highest counter, which indicates the highest frequency of past occurrence. If the current node is a leaf, the LZ predictor makes no prediction.

LZ-Prefix and PPM.

Bhattacharya and Das propose a heuristic modification to the construction of the LZ tree [5], as well as a way of using the modified tree to predict the most likely cells that contain the user so as to minimize paging costs to locate that user.

As we mention above, not every substring in L forms a node s_i in the LZ parsing tree. In particular, substrings that cross boundaries of the s_i, $0 < i \leq m$, are missed. Further, previous LZ-based predictors take into account only the occurrence statistics for the prefixes of the leaves. To overcome this, they proposed the following modification. When a new leaf is created for s_i, all the proper suffixes of s_i (i.e., all the suffixes not including s_i itself) are also inserted into the tree. If a node representing a suffix does not exist, it is created; and the occurrence counter for every prefix of every suffix is incremented.

Example. Suppose the current leaf is $s_m = bde$ and the string de is one that crosses boundaries of existing s_i for $1 \leq i < m$. Thus de has not occurred as a prefix or a suffix of any s_i, $0 < i < m$. The set of proper suffixes of s_m is $S_m = \{\gamma, e, de\}$, and since there is no node representing the substring for de, it is created. Then the occurrence frequency is incremented for the root labeled γ, the first-level children b and d, and the new leaf node de. Figure 3 shows the tree after this transformation, which we call "LZ+prefix" or "LZP" for short.

We observe that this heuristic only discovers substrings that lie within a leaf string. Nonetheless, at this point it is possible to use the modified LZ tree and apply one of the existing prediction heuristics, e.g., use Eq. 2 and the Vitter-Krishnan method. Indeed, we include the LZP predictor in our comparison.

Bhattacharya and Das [5] propose a further heuristic that uses the LZP tree for prediction. This second heuristic is based on the Prediction by Partial Match (PPM) algorithm for text compression [13]. (The PPM algorithm essentially

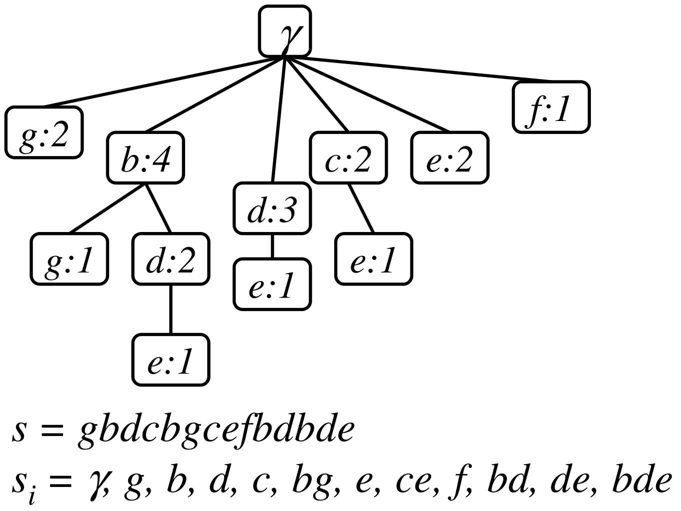


Figure 3. Example LZP parsing tree

attempts to "blend" the predictions of several $O(k)$ Markov predictors; we do not describe it here in detail.) Given a leaf string s_m, construct its set of proper suffixes S_m and update the LZP tree as described above. Then, for each such suffix, the heuristic considers the subtree rooted at the suffix and finds all the paths in this subtree originating from this subtree root. The PPM algorithm is then applied to this set of paths. PPM first computes the predicted probability of each path, and then uses these probabilities to compute the most probable next symbol(s) based on their weights (number of occurrences) in the path. We include the "LZ+prefix+PPM" predictor, nicknamed "LeZi" [5], in our comparison.

C. Breaking ties

In our descriptions of the predictors, above, we indicate that each predicts the symbol with the highest probability of occurring next, given the current state of its data structure (Markov matrix or LZ tree). It is quite possible, though, for there to be a tie for first place, that is, several symbols with equal probability and none with higher probability. None of the literature indicates how to break a tie, yet our implementations must make a specific choice. We implemented 3 different tie-break methods:

- *First added:* among the symbols that tie, we predict the first one added to the data structure, that is, the first one seen in the location history.
- *Most recently added:* among the symbols that tie, we predict the one that was most recently added to the data structure.
- *Most recent:* among the symbols that tie, we predict the one most recently seen in the history.

In Figure 4, we show the ratio of the number of ties and the number of predictions, using a cumulative distribution function (CDF) across all users having more than 1000 movements for different predictors (the O(2) Markov fallback predictor will be described later). Fortunately, Figure 4 shows that most of the users had few ties, less than about 10% of all predictions.

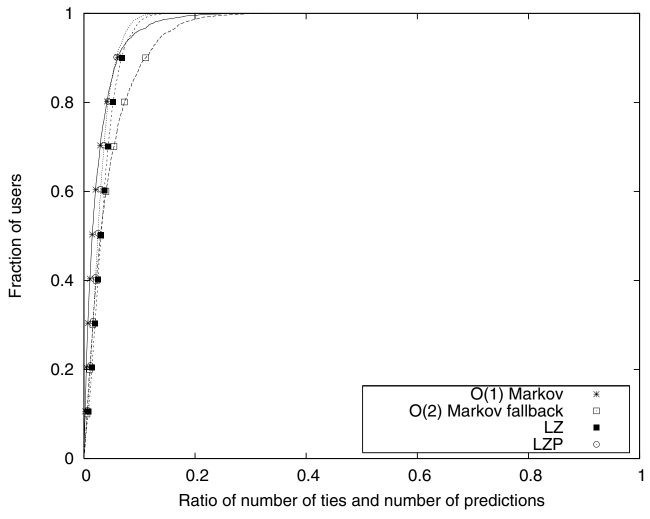


Figure 4. Ratio of number of ties and the number of predictions

Our experiments showed that the results were not significantly affected by the choice of a tie-breaking method. We used the "first added" method throughout the results below, and do not consider tie-breakers further.

D. Entropy metric

We believe that there are some intrinsic characteristics of a trace that ultimately determine its predictability and hence the performance of different predictors. In the results section, we compare the accuracy metric with the entropy metric, for each user, on several predictors.

In general, the entropy $H(X)$ of a discrete random variable X is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} P(x) \log_2 P(x) \tag{3}$$

where $\mathcal{X}$ is the set of all possible of values of x.

In this paper, we compute the entropy of a given user under the $O(k)$ Markov predictor. We obtain the probabilities $P(x)$ from the data structures constructed by that predictor on that user's trace.

If C is the set containing all the context strings encountered in parsing a location history L, then the conditional entropy is

$$H(X|C) = -\sum_{c \in \mathcal{C}} \frac{N(c,L)}{n-k+1} \sum_{a \in \mathcal{A}} P(x=a|c) \log_2 P(x=a|c) \tag{4}$$

E. Accuracy metric

During an on-line scan of the location history, the predictor is given a chance to predict each location. There are three possible outcomes for this prediction, when compared to the actual location:

- The predictor correctly identified the location of the next move.
- The predictor incorrectly identified the location of the next move.
- The predictor returned "no prediction."

All predictors encounter situations in which they are unable to make a prediction; in particular, all realistic predictors will have no prediction for the first location of each user trace.

We define the *accuracy* of a predictor for a particular user to be the fraction of locations for which the predictor correctly identified the next move. Thus, "no prediction" is counted as an incorrect prediction. In the future we plan to examine other metrics that can better distinguish the two forms of incorrect prediction (there may be some applications that may prefer no prediction to a wild guess). For now, this metric best reflects the design of predictors common in the literature.

F. Data collection

We have been monitoring usage on the Wi-Fi network at Dartmouth College since installation began in April 2001. Installation was largely complete by June 2001, and as of March 2003 there are 543 access points providing 11 Mbps coverage to the entire campus. Although there was no specific effort to cover outdoor spaces, the campus is compact and the interior APs tend to cover most outdoor spaces. The wireless population is growing fast. As of May 2003 there are between 2,500 and 3,500 users active in any given day.

The access points transmit a "syslog" message every time a client card associated, re-associated, or disassociated; the message contains the unique MAC address of the client card. Although a given card might be used in multiple devices or a device used by multiple people, in this paper we think of the wireless card as a "network user" and thus the term "user" refers to a wireless card.

We have a nearly continuous, two-year record of these syslog messages from April 2001 through March 2003. Our data has some brief "holes" when the campus experienced a power failure, or when a central syslog daemon apparently hung up. Also, since syslog uses UDP it is possible that some messages were lost or misordered. In March 2002, we installed two redundant servers to record the data, so that holes in one of the servers can be filled by the data recorded by the other server.

For more information about Dartmouth's network and our data collection, see our previous study [14] [15].

After we cleaned the data of some glitches, and merged the data from two servers (where available), we extracted user traces from the data. Each user's trace is a series of locations, that is, access-point names. We introduced the special location "OFF" to represent the user's departure from the network (which occurs when the user turns off their computer or their wireless card, or moves out of range of all access points). The traces varied widely in length (number of locations in the sequence), with a median of 494 and a maximum of 188,479; most are shown in Figure 5. Users with longer traces were either more active (using their card more), more mobile (thus changing access points more often), or used the network for a longer period (some users have been on the network since

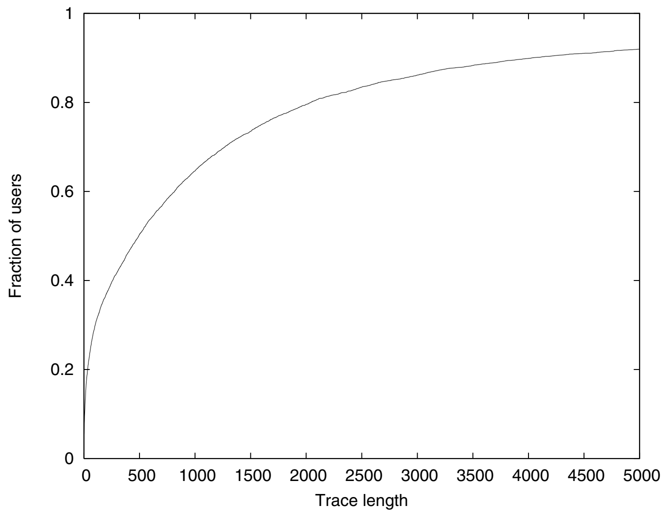


Figure 5. Length of user traces

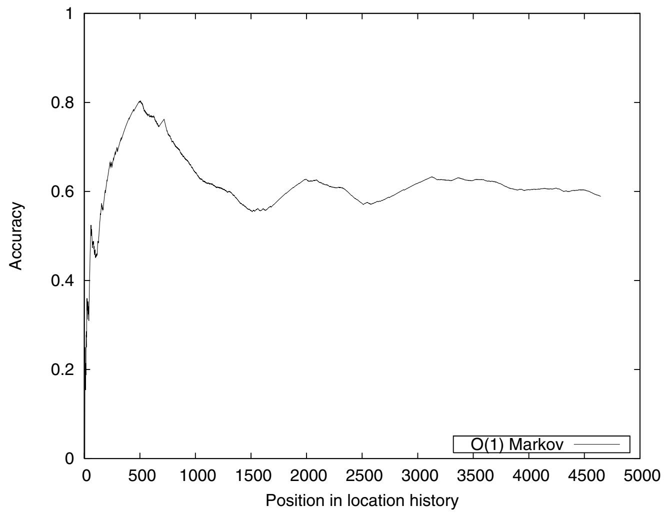


Figure 6. Prediction accuracy for a sample user

April 2001, and some others have only recently arrived on campus).

G. Evaluation experiments

To evaluate a predictor, we run each user's trace independently through our implementation of that predictor. For each location in the trace, the predictor updates its data structure and then makes a prediction about the next location. Our experiments framework tracks the accuracy of the predictor.

III. Results

We evaluated the location predictors using our Wi-Fi mobility data. In this section we examine the results.

A. Markov

Recall that accuracy is defined as the number of correct predictions divided by the number of attempts (moves). As we step through the locations in a single trace, attempting to predict each location in sequence, we can examine the accuracy up to that point in the trace. Figure 6 shows how the accuracy metric varies over the course of one user's history, using the $O(1)$ Markov predictor. Ultimately, for comparison purposes, we define the accuracy over the entire trace, the rightmost value in this plot. In subsequent graphs we use this overall accuracy as the performance metric.

Of course, we have several thousand users and the predictor was more successful on some traces than on others. In Figure 7 we display the accuracy of the $O(1)$ Markov predictor in CDF curves, one for each of three groups of traces: short, medium, and long traces. It is obvious that curves "to the right" are better, because for a given fraction of users, the right curve has a higher prediction accuracy. The predictor was clearly unsuccessful on most short traces (100 or fewer moves), because its curve is far to the left. Ultimately, we found that all predictors fared poorly on traces with fewer than 1000 moves, and we restrict the remainder of our comparison

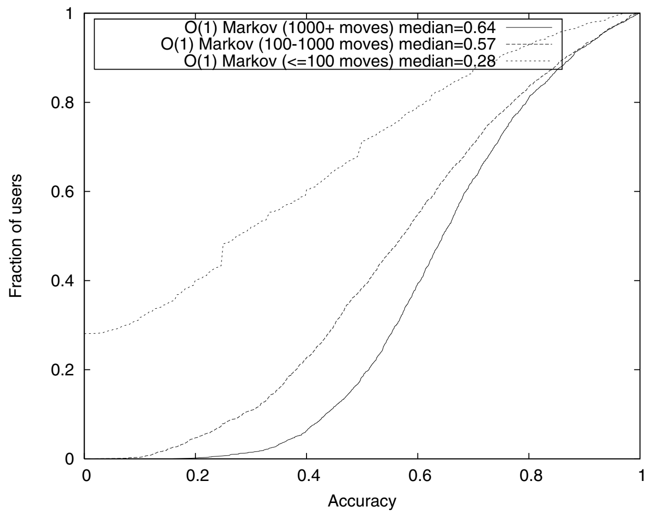


Figure 7. Accuracy of $O(1)$ Markov predictor

to traces of more than 1000 moves. There were 2,195 such users, out of 6,202 total users traced.

After such strong evidence that the predictor's accuracy was better, in general, for users with longer traces, it is tempting to suggest that accuracy is somewhat correlated with trace length. Figure 8 also shows there is a slight trend that longer traces have higher prediction accuracy, although it is not a linear dependence.

Intuitively, it should help to use more context in each attempt to predict the next location. In Figure 9 we compare the accuracy of $O(1)$ Markov with that of $O(2)$, $O(3)$, and $O(4)$ Markov predictors. As an aside, we include an "order 0" Markov predictor, in which no context is used in the prediction of each move. This predictor simply predicts the most frequent location seen so far in that user's history. Although it represents a degenerate form of Markov prediction, it helps to show the benefit of context in the $O(1)$ predictor.

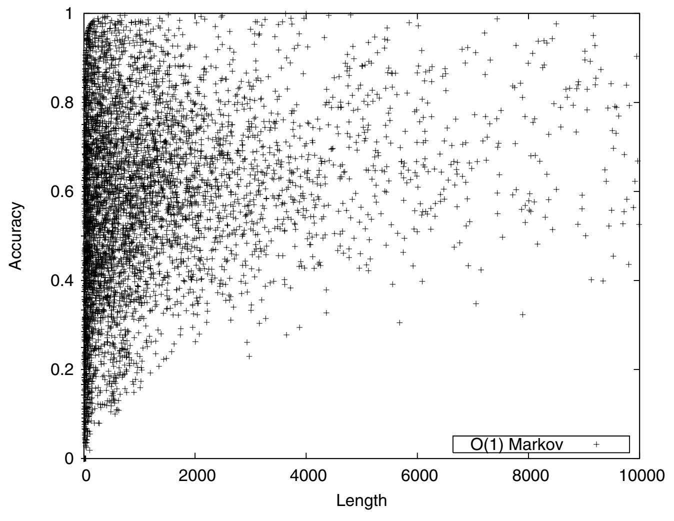


Figure 8. Correlating accuracy with trace length

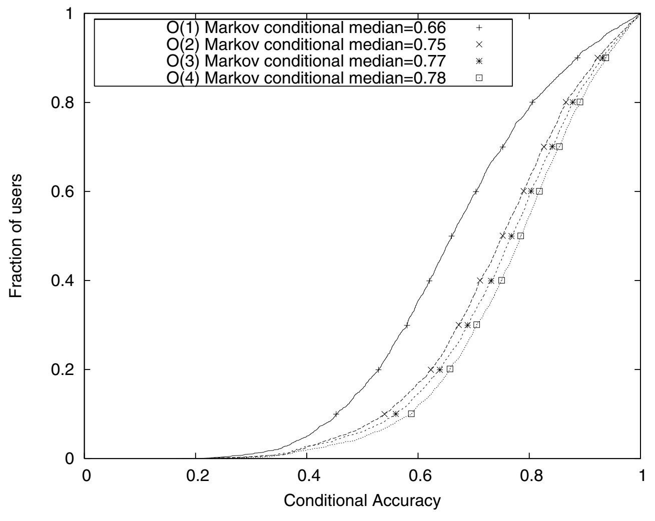


Figure 10. Conditional accuracy metric

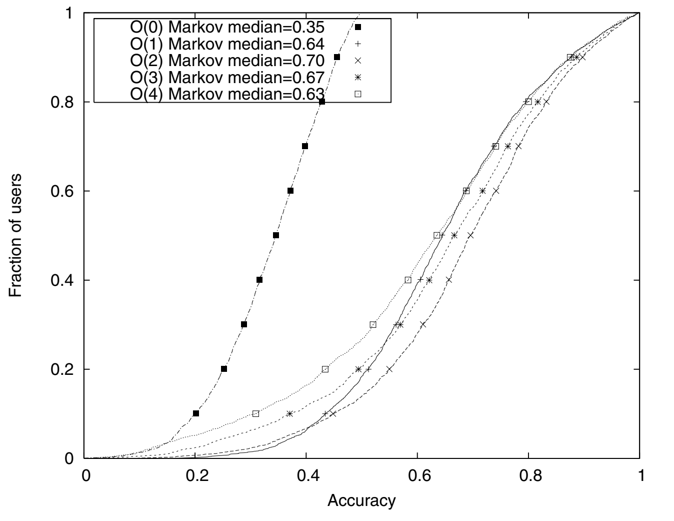


Figure 9. Comparing Markov predictors

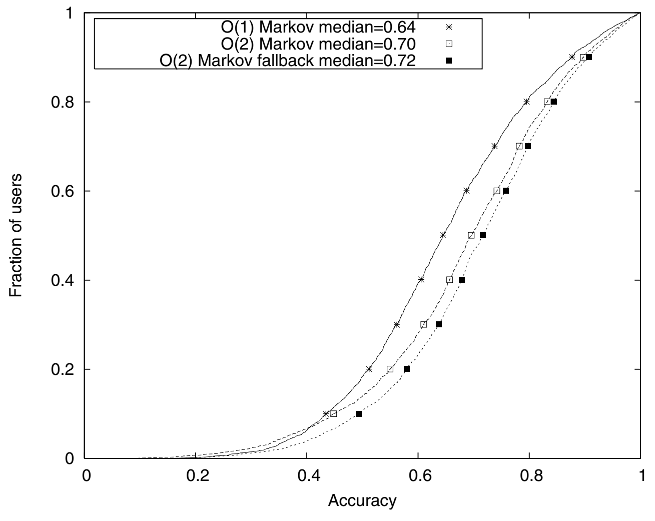


Figure 11. Markov predictors with fallback

Not surprisingly, $O(2)$ generally outpredicted $O(1)$, and its curve is further to the right. The high-order $O(3)$ and $O(4)$ predictors were, however, worse than $O(2)$. Although these predictors use more information in the prediction process, they are also more likely to encounter a context (a three- or four-location string) that has not been seen before, and thus be unable to make a prediction. A missing prediction is not a correct prediction, and these unpredicted moves bring down the accuracy of the higher-order predictors. In Figure 10 we display the *conditional accuracy* of these same predictors: the number of correct predictions divided by the number of predictions. In this metric, we ignore unpredicted moves, and it becomes clear that longer context strings did lead to better prediction, where possible, although with diminishing returns.

Returning to our original accuracy metric, we now consider a meta-predictor based on the Markov predictor family. Figure 11 displays the performance of the $O(2)$ Markov predictor with "fallback," which used the results of the $O(2)$ predictor when it made a prediction, or the $O(1)$ predictor if the $O(2)$ predictor had no prediction. In general, the $O(k)$ fallback predictor recursively used the result of the $O(k-1)$ predictor (with $k=0$ as the base of the recursion) whenever it encountered an unknown context. Fallback indeed helped to improve the $O(2)$ Markov predictor's performance. $O(3)$ and $O(4)$ Markov also improved with fallback, but the results are nearly identical to the $O(2)$ Markov with fallback and are thus not shown. Markov predictors with fallback gain the advantage of the deeper context but without losing accuracy by failing to predict in unknown situations.

In the plots so far we examine the performance of $O(k)$ Markov predictors that used the most recent k locations in making a prediction, weighting the potential next locations by their frequency of occurrence in the past. An alternative approach assigns weights according to the *recency* of occurrence

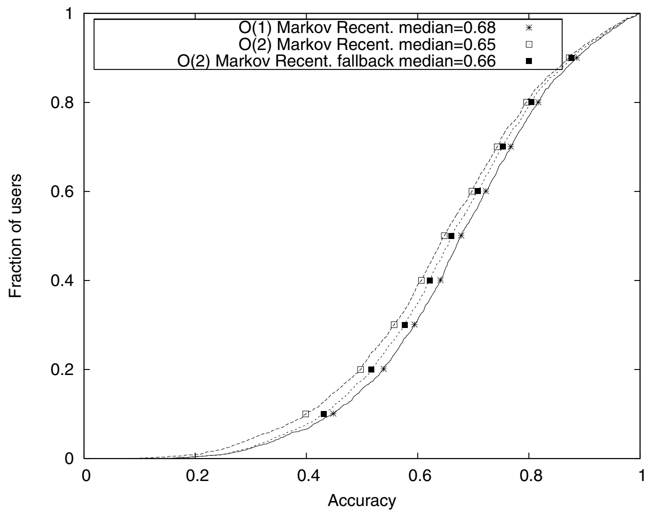


Figure 12. Markov using "most recent"

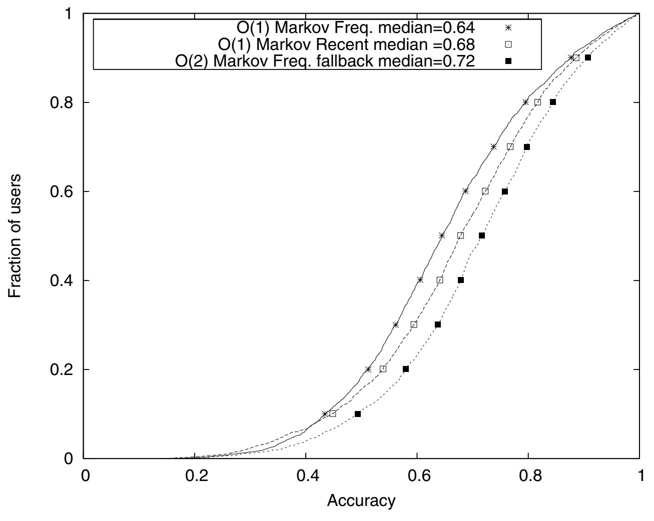


Figure 13. Markov: "most recent" vs. "most frequent"

in the past; thus, one transition (the most recent seen for this context) has weight 1 and the others have weight 0. Observe that the Order-k Markov prediction using the most-frequent location in general takes $O(n^{k+1})$ space, while using the most-recent location it takes only $O(n^k)$ space, a potentially significant decrease in storage. Figure 12 shows the quality of Markov predictors based on this approach. Curiously, here $O(2)$ did worse than $O(1)$, although fallback made up some of the difference. We are still exploring the reason behind this curious difference.

In Figure 13 we compare the recency-weighted approach with the original frequency-weighted approach. The best recency-weighted Markov predictor, $O(1)$, was better than the corresponding frequency-weighted predictor. This result implies that recent history was a better predictor of the immediate future than was an accumulated probability model, when considering only the current location. On the other hand, recall from Figure 12 that among the recency-weighted predictors the extra context of $O(2)$ lowered prediction accuracy. Thus, among $O(2)$ predictors, the frequency-weighted approach beats the recency-weighted approach (not shown in Figure 13). Ultimately, the $O(2)$ frequency-weighted Markov predictor with fallback has the best outcome.

Some of our user traces span a period of hours or days, but some span weeks or months. Clearly it is possible for a user's mobility patterns to change over time; for example, a student may move to a new dormitory, or attend different classes. The transition weights constructed by the frequency-weighted Markov model may become ineffective after such a change; for users with a long history, these weights will adapt too slowly. In another series of experiments (not shown), we added an exponential decay to the frequency weights so that more recent locations have a larger impact, but we saw little improvement in the quality of the predictors. We need to study this option further.

B. *LZ-based*

We first consider the LZ predictor, shown in Figure 14. Since LZ makes no prediction whenever the current context string leads to a leaf in the tree, the plot includes two LZ variants. As an alternative to no prediction (the first curve), we can use the statistics at the root (second curve) to make a prediction based on the probability of each location. That is, when the predictor encounters a leaf, it behaves as if it is at the root and simply predicts the most frequently seen child of the root. Given our accuracy metric, it is always better to make some prediction than no prediction, and indeed in this case the accuracy improved significantly. An even better approach (third curve) is to fallback to progressively shorter substrings of the current context, much as we did with the Markov predictors, until a substring leads to a non-leaf node from which we can make a prediction. This fallback ability is critical to allow prediction to occur while the tree grows, since the trace often leads to a leaf just before adding a new leaf.

Later, Bhattacharya and Das [5] proposed two extensions to the LZ predictor. Figure 15 displays the performance of the first extension, LZP. Once again, this predictor (as defined) makes no prediction when the context leads to a leaf, and once again it helped to use statistics at the root, or (better) fallback. The second extension produces a LZ+prefix+PPM predictor nicknamed "LeZi." Figure 16 compares the performance of LZ with LZP and LeZi, showing that each extension did improve the accuracy of the LZ predictor substantially.

When we compare the best variant of each LZ form, in Figure 17, it becomes clear that the simple addition of our fallback technique to the LZ predictor did just as well as the prefix and PPM extensions combined. (LeZi automatically includes fallback by adding suffix substrings into the tree, so there is no fallback variant.) LZ with fallback is a much simpler predictor than LZP or LeZi, and since the accuracy is similar (and as we show below, the LZ data structure was

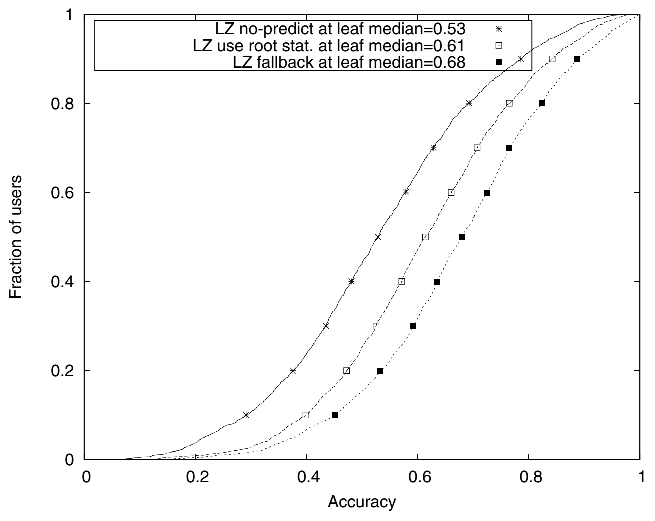


Figure 14. LZ predictors

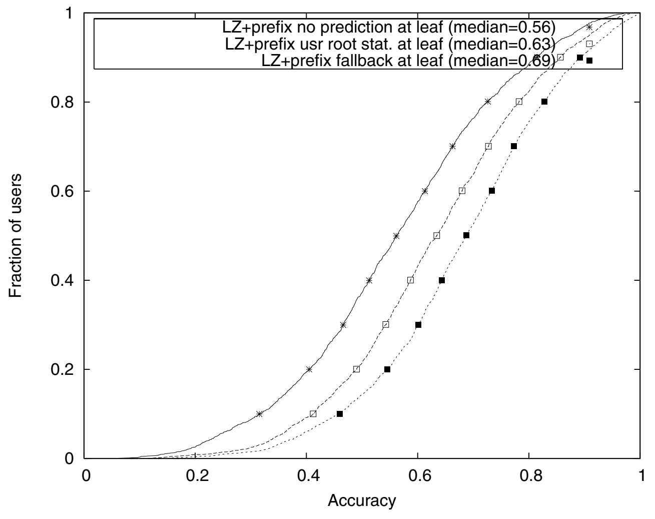


Figure 15. LZP predictors

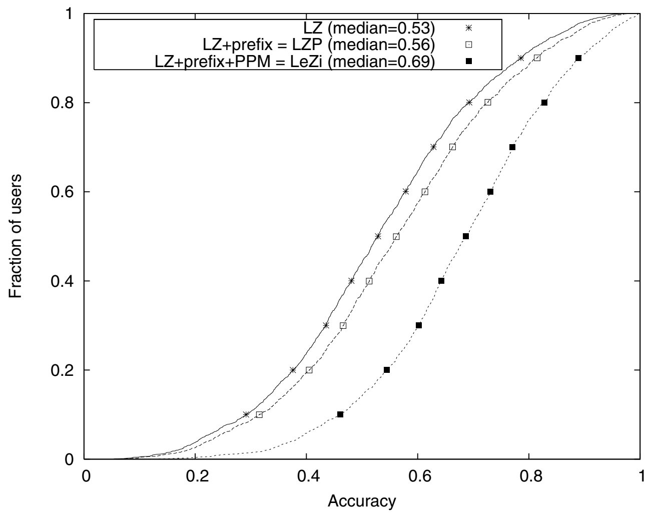


Figure 16. LZ, LZP, LeZi predictors

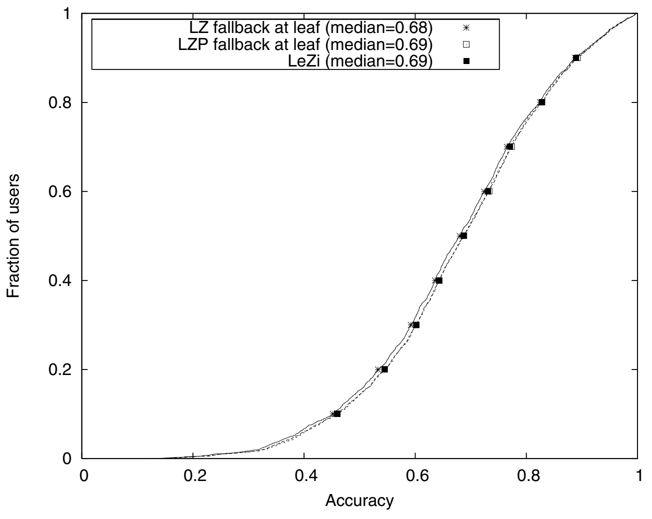


Figure 17. LZ predictors with fallback

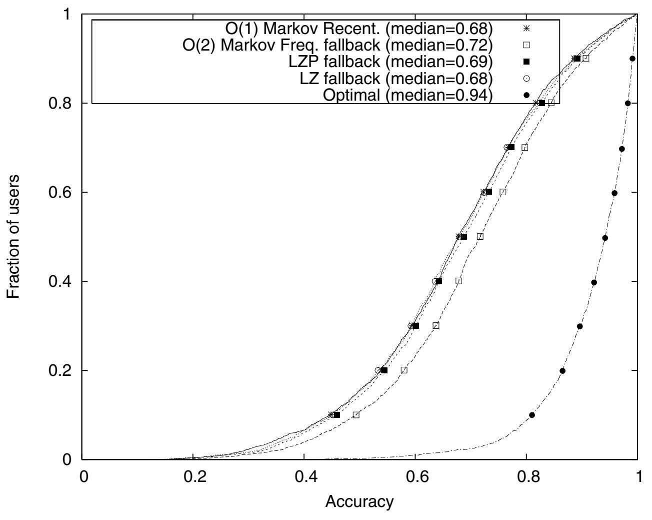


Figure 18. The best predictors, compared

smaller) we recommend LZ with fallback.

C. Overall

We compare the best Markov predictors with the best LZ predictors in Figure 18. It is difficult to distinguish the LZ family from the recency-based $O(1)$ Markov, which all seem to have performed equally well. The $O(2)$ Markov predictor, with fallback to $O(1)$ whenever it had no prediction, was the best overall. It is striking that the extra complexity, and the theoretical aesthetics, of the LZ predictors apparently gave them no advantage.

We include an "Optimal" curve in Figure 18, as a simple upper bound on the performance of history-based location predictors. In our definition, the "optimal" predictor can accurately predict the next location, except when the current location has never been seen before. Although it should be possible to define a tighter, more meaningful upper bound for domain-independent predictors like those we consider here, it

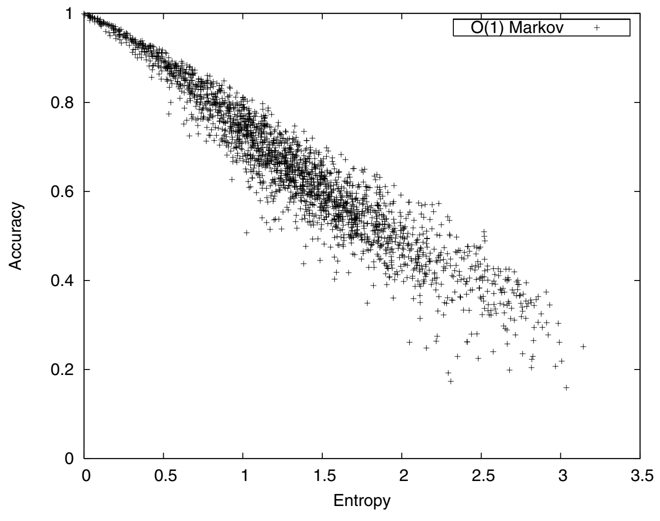


Figure 19. Correlating accuracy with entropy

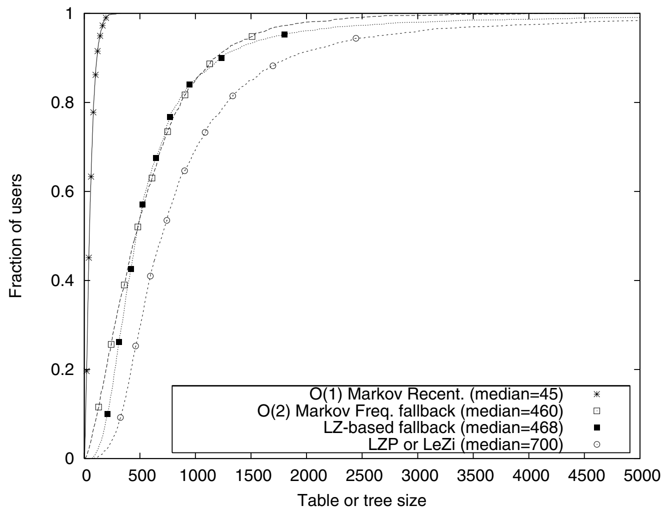


Figure 20. Distribution of final table sizes

seems clear that there is room for better location-prediction algorithms in the future. It may be that some user traces are simply less predictable than others. In Figure 19 we compare the entropy of each user, based on the probability table built by the $O(1)$ Markov predictor, with the accuracy of the $O(1)$ predictor. The correlation is striking, and indeed the correlation coefficient is -0.95 (a coefficient of 1.0 or -1.0 represents perfect correlation). This strong correlation indicates that some users with high entropy are doomed to poor predictability.

Other metrics. Of course, prediction accuracy is not the only metric by which to compare location predictors. In Figure 20 we show the size of the predictors' data structures, at the end of simulating each user. As with the accuracy metric, we plot the table size for each predictor as a distribution across all users. For Markov predictors, we define the size to be the number of entries in the (sparse) transition matrix, except for the recency-based $O(1)$ Markov, which simply needs to record the location of the latest transition for each location ever visited. For LZ predictors, we define the size to be the number of tree nodes. (Since the size of each table entry or tree node is implementation dependent, we do not measure table sizes in bytes.)

Clearly the recency-based $O(1)$ Markov had the simplest and smallest table, by far. In second place are the $O(2)$ Markov and LZ predictors. The LZP and LeZi predictors used by far the most space.

Although the medians of $O(2)$ Markov and LZ predictors appear to be similar, on a closer examination it is clear that the LZ predictor has a much higher maximum. Indeed, this plot is truncated at the right side; for one user, LZ used 17,374 nodes. LeZi used as many as 23,361 nodes! The Markov predictor, on the other hand, never used more than 5,000 table entries.

Computational complexity is another important metric, in some applications; we leave the asymptotic analysis to the theoretical literature. Furthermore, we have not yet evaluated the running time of these predictors on our data (e.g., average microseconds per prediction), as we have not yet tuned our implementation. Nonetheless, it is intuitively clear that the simpler Markov predictors are likely to be faster than the LZ predictors.

IV. Conclusions

In this paper we compared two major families of domain-independent on-line location predictors (Markov and LZ) by measuring their accuracy when applied to the mobility traces we collected from 6,202 users of Dartmouth College's wireless network.

Most of the traces in our collection were short, less than 1000 movements, and all predictors fared poorly on most of those users. These predictors typically required a fair amount of history to initialize their probability tables to a useful state. The rest of our results are based exclusively on the long-trace users.

In general, the simple low-order Markov predictors worked as well or better than the more complex compression-based predictors, and better than high-order Markov predictors. In particular, $O(3)$ (or above) Markov predictors did not improve over $O(2)$ Markov, and indeed reduced accuracy by failing to make predictions much of the time.

Most of the predictors, as defined in the literature, fail to make a prediction when faced with a context (recent history) that has never been encountered in the full user's history. We found it was simple to add "fallback" to each predictor, allowing the predictor to use shorter and shorter context until it was able to make a prediction, and that this fallback often improved the predictor's accuracy substantially.

In particular, $O(2)$ Markov with fallback beat or equaled all of the LZ-based predictors, even though the latter have better theoretical asymptotic behavior [7]. We found $O(2)$ Markov with fallback to be the best overall predictor, was simple to implement, had relatively small table size, and had the best overall accuracy.

We experimented with a simple alternative to the frequency-based approach to Markov predictors, using recency (probability 1 for most recent, 0 for all other) to define the transition matrix. Although this recency approach was best among $O(1)$ Markov predictors, it was worst among $O(2)$ Markov predictors, and we are still investigating the underlying reason.

We found most of the literature defining these predictors to be remarkably insufficient at defining the predictors for implementation. In particular, none defined how the predictor should behave in the case of a tie, that is, when there was more than one location with the same most-likely probability. We investigated a variety of tie-breaking schemes within the Markov predictors, but found that the accuracy distribution was not sensitive to the choice.

Since some of our user traces extend over weeks or months, it is possible that the user's mobility patterns do change over time. All of our predictors assume the probability distributions are stable. We briefly experimented with extensions to the Markov predictors that "age" the probability tables so that more recent movements have more weight in computing the probability, but the accuracy distributions did not seem significantly affected. We need to study this issue further.

We examined the original LZ predictor as well as two extensions, prefix and PPM. LZ with both extensions is known as LeZi. We found that both extensions did improve the LZ predictor's accuracy, but that the simple addition of fallback to LZ did just as well, was much simpler, and had a much smaller data structure. To be fair, PPM tries to do more than we require, to predict the future path (not just the next move).

We stress that all of our conclusions are based on our observations of the predictors operating on over 2000 users, and in particular whether a given predictor's accuracy distribution seems better than another predictor's accuracy distribution. For an individual user the outcome may be quite different than in our conclusion. We plan to study the characteristics of individual users that lead some to be best served by one predictor and some to be best served by another predictor.

There was a large gap between the predictor's accuracy distribution and the "optimal" accuracy bound, indicating that there is substantial room for improvement in location predictors. On the other hand, our optimal bound may be overly optimistic for realistic predictors, since it assumes that a predictor will predict accurately whenever the device is at a location it has visited before. We suspect that domain-specific predictors will be necessary to come anywhere close to this bound.

Overall, the best predictors had an accuracy of about 65–72% for the median user. On the other hand, the accuracy varied widely around that median. Some applications may work well with such performance, but many applications will need more accurate predictors; we encourage further research into better predictors.

We continue to collect syslog data, extending and expanding our collection of user traces. We plan to evaluate domain-specific predictors, develop new predictors, and develop new accuracy metrics that better suit the way applications use location predictors. We also plan to predict the location along with the time in the future.

ACKNOWLEDGEMENTS

The Dartmouth authors thank DoCoMo USA Labs, for their funding of this research, and Cisco Systems, for their funding of the data-collection effort. We thank the staff of Computer Science and Computing Services at Dartmouth College for their assistance in collecting the data.

REFERENCES

[1] David A. Levine, Ian F. Akyildiz, and Mahmoud Naghshineh, "The shadow cluster concept for resource allocation and call admission in ATM-based wireless networks," in *Mobile Computing and Networking*, 1995, pp. 142–150.

[2] William Su and Mario Gerla, "Bandwidth allocation strategies for wireless ATM networks using predictive reservation," in *Proceedings of Global Telecommunications Conference (IEEE Globecom)*, November 1998, vol. 4, pp. 2245–2250.

[3] George Liu and Gerald Maguire Jr., "A class of mobile motion prediction algorithms for wireless mobile computing and communications," *ACM/Baltzer Mobile Networks and Applications (MONET)*, vol. 1, no. 2, pp. 113–121, 1996.

[4] Sajal K. Das and Sanjoy K. Sen, "Adaptive location prediction strategies based on a hierarchical network model in a cellular mobile environment," *The Computer Journal*, vol. 42, no. 6, pp. 473–486, 1999.

[5] Amiya Bhattacharya and Sajal K. Das, "LeZi-Update: An information-theoretic approach to track mobile users in PCS networks," *ACM/Kluwer Wireless Networks*, vol. 8, no. 2-3, pp. 121–135, March–May 2002.

[6] Fei Yu and Victor C. M. Leung, "Mobility-based predictive call admission control and bandwidth reservation in wireless cellular networks," *Computer Networks*, vol. 38, no. 5, pp. 577–589, 2002.

[7] Christine Cheng, Ravi Jain, and Eric van den Berg, "Location prediction algorithms for mobile wireless systems," in *Handbook of Wireless Internet*, M. Illyas and B. Furht, Eds. CRC Press, 2003.

[8] Sajal K. Das, Diane J. Cook, Amiya Bhattacharya, Edwin Heierman, and Tze-Yun Lin, "The role of prediction algorithms in the MavHome smart home architecture," *IEEE Wireless Communications*, vol. 9, no. 6, pp. 77–84, 2002.

[9] Jeffrey Scott Vitter and P. Krishnan, "Optimal prefetching via data compression," *Journal of the ACM*, vol. 43, no. 5, pp. 771–793, 1996.

[10] Jacob Ziv and Abraham Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, Sept. 1978.

[11] P. Krishnan and Jeffrey Scott Vitter, "Optimal prediction for prefetching in the worst case," *SIAM Journal on Computing*, vol. 27, no. 6, pp. 1617–1636, 1998.

[12] Meir Feder, Neri Merhav, and Michael Gutman, "Universal prediction of individual sequences," *IEEE Transactions on Information Theory*, vol. 38, no. 4, pp. 1258–1270, July 1992.

[13] Timothy C. Bell, John G. Cleary, and Ian H. Witten, *Text Compression*, Prentice Hall, 1990.

[14] David Kotz and Kobby Essien, "Analysis of a campus-wide wireless network," in *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking*, September 2002, pp. 107–118, Revised and corrected as Dartmouth CS Technical Report TR2002-432.

[15] David Kotz and Kobby Essien, "Analysis of a campus-wide wireless network," *ACM Mobile Networks and Applications (MONET)*, 2003, Accepted for publication.